```
import torch
import torch.nn.functional as F             # Para usar linear.
import torchvision as tv


class RBM( torch.nn.Module):
    def __init__( _, vsize, hsize, CD_k=1):
        super().__init__()
        _.W  = torch.nn.Parameter( torch.randn( hsize, vsize)*1e-2)
        _.bv = torch.nn.Parameter( torch.randn( vsize)*1e-2) # Bias V.
        _.bh = torch.nn.Parameter( torch.randn( hsize)*1e-2) # Bias H.
        _.k = CD_k                          # Divergencia Contrastiva.

    def sample_h( _, v):
        prob_h = torch.sigmoid( F.linear( v, _.W, _.bh))
        samp_h = torch.bernoulli( prob_h)
        return prob_h, samp_h

    def sample_v( _, h):
        prob_v = torch.sigmoid( F.linear( h, _.W.t(), _.bv))
        samp_v = torch.bernoulli( prob_v)
        return prob_v, samp_v

    def forward( _, v):
        vs = v
        for i in range( _.k):
            hp, hs = _.sample_h( vs)
            vp, vs = _.sample_v( hs)
        return v, vs

    def free_energy( _, v):
        v_bv = v.mv( _.bv)          # Multiplica matriz por vector.
        hlin = torch.clamp( F.linear( v, _.W, _.bh), -80, 80)
        slog = hlin.exp().add(1).log().sum(1)
        return ( -slog - v_bv).mean()


if __name__ == '__main__':          # Para poder importar RBM.
    T = 20                          # Cant de epocas.
    B = 64                          # Mini-lotes.
                                    # trn/tst-data/load como antes.
    N = 28*28                       # Cant de entradas.
    M = 64                          # Cant de features.
    C = 10                          # Cant de clases de salida.

    rbmfd = RBM( N, M)                              # Feature Detector
    optim = torch.optim.SGD( rbmfd.parameters(), 0.1)
    for t in range(T):
        E = 0
        for images, labels in trn_load:
            optim.zero_grad()
            data = images.view( -1, N)
            v0, vk = rbmfd( data)
            loss = rbmfd.free_energy(v0) - rbmfd.free_energy(vk)
            loss.backward()
            optim.step()
            E += loss.item()
        print( t, E)

    lincl = torch.nn.Linear( M, C)                  # Linear Classifier
    optim = torch.optim.SGD( lincl.parameters(), 0.1)
    costf = torch.nn.CrossEntropyLoss()
    for t in range(T):
        E = 0
        for images, labels in trn_load:
            optim.zero_grad()
            v = images.view( -1, N)
            hp, hs = rbmfd.sample_h( v)
            cp = lincl( hp)
            error = costf( cp, labels)
            error.backward()
            optim.step()
            E += error.item()
        print( t, E)

    right = 0
    total = 0
    with torch.no_grad():
        for images, labels in tst_load:
            v = images.view( -1, N)
            hp, hs = rbmfd.sample_h( v)
            cp = lincl( hp)
            right += (cp.argmax(dim=1)==labels).sum().item()
            total += len(labels)
    print("Accuracy:", right/total)


# Ver: http://deeplearning.net/tutorial/rbm.html
```